Code Reviews Theory and Practice

Gordon Letwin

Microsoft Corporation

November 5, 1984

ABSTRACT

This memo discusses the purpose and practice of a code review, and reviews the approaches that found to be most effective at Microsoft.

Introduction

The DOS4 project is the first OS effort to make extensive use of code reviews. We have found that good code reviews improves the code quality, not only after the review but before, as well. When we started formal code reviews we discovered that there were many misconceptions about the purpose and process of a code review. We also discovered that to get good results from a review both the reviewer and author must put effort into the process.

This memo discusses the purpose and practice of a code review, and reviews the approaches that we have found to be most effective.

Code Reviews - What and Why

A code review is the reading of one person's code by another. This is not normally done as face-to-face meeting. instead, the author sends the code to the reviewer and the reviewer returns a list of comments. Frequently, the reviewer illustrates the comments by editing the code itself. In this case, the edited code is returned and the author 'diffs' it against the original to extract the changes.

There are four key goals for a code review:

1. **Guarantee Readability, Understandability and Maintainability**

   This is the primary goal of the code review and is simultaneously the hardest and easiest to achieve. Simply put, can the reviewer pick up the code and immediately start understanding it? Within the first minute or so the reader should understand the purpose of the code, its major inputs, outputs, data structures and algorithms. The reviewer should, almost immediately, gain a clear grasp of just what this package does, to whom, and how.

   At a lower level the code should, by means of comments and lexical layout, literally explain itself. The partitioning of the work into subroutines should be clear and unambiguous. At any given point in any subroutine the reader should be able to quickly understand what is happening, what the control flow is, what the register and local variable contents are, etc.

   While reading the code, the reviewer should play the role of a future maintainer or debugger. At each point in each program, can the hypothetical maintainer understand which values are valid, the meaning of those values, and which registers, variables and subroutines may be used?

   I say that this is the easiest goal of a review because the goal of programming is not to cause a CPU to produce a desired result. Instead, the goal of programming is to encode the functioning of a correct and well understood algorithm. This encoding must meet two requirements: it must be transformable, via compiler or assembler, into functioning machine instructions, and it must be clearly and readily understandable to the engineers who will be working with it, not only the original engineer, but the reviewer and maintenance/modification engineers, as well.

   The key factor to remember, both as programmer and reviewer, is that what we are doing is engineering, not writting puzzles. A reader is expected to 'figure out' a puzzle, but a knowledgeable reader should never have to 'figure out' good code, whether that 'figuring out' be 'playing compiler/assembler' because of poor lexical style (indentation, parenthesis) or whether it be a full scale 'safari into the unknown'.

2. **Check for adherence to Design and Coding Standards**

   This is the second-most important goal of a code review. This step is relatively mechanical but never the less very important since the code review is the only mechanism in the design cycle which can locate problems here and get them corrected.

3. **Review the Design for Appropriateness**

   The reviewer evaluates the algorithms to insure that they accomplish the required task and will be able to meet performance criterion. For example, a bubble sort algorithm used to sort an assembler symbol table would not be appropriate.

   This is the least important of the code review goals because it should be merely a formality and a sanity-check; the author should have proven his approach appropriate long before a code review.

4. **Look for Bugs and 'Clumsies'**

   Finding bugs in a program is generally considered the prime goal of a code review. It is not and in fact appears low on this priority list. Although not of prime importance, finding bugs is certainly a valuable fallout from the code review. The time saved by bug detection during the review generally 'pays' for the code review immediately.

   A 'clumsy' is a sequence of code which, although correct, may be rewritten to optimize either speed or size or, in some cases, both. The degree of zeal with which clumsys are pursued depends upon the particular section of code. If the code is infrequently used and is not size critical then readability and understandability are the key concerns, not bytes and microseconds. Space and time-critical code, especially inner loops, should be scanned carefully for clumsys. Of course, the degree of 'trickiness' of the proposed replacement sequence must correlate to the space/time payback. Naturally, any truly tricky sequences must be very clearly documented.

   Most bugs are discovered while pursuing goal #1 above. In addition, a good reviewer knows that most, if not all, programs contain bugs. Coupling that knowledge with an understanding of the prime bug hiding spots in typical code, the reviewer should be able to flush out a few of them.

Key Stumbling Blocks in Reviews

The #1 stumbling block to the effectiveness of code reviews is the programmer's ego involvement in the code. Its natural for a programmer to see the code as 'his baby' and to take any alterations to it as criticisms. These perceived 'criticisms' are often taken as attacks on the programmer rather than as reasonable suggestions for improving the program. It is natural... but its wrong. No programmer is born with the skills to write code, good or otherwise. We have all learned what we know, somewhere, somehow and none of us has learned all there is to know. The key to getting good results from a code review is to see it not as personal criticism but as a learning situation.

It is easier to accept 'learning' in school because of the formal relationship between the instructor, who knows everything, and the student, who knows nothing (at least, that's how they explained it to me...). Of course, this is not the only possible way to learn things. The person reviewing your code may not especially know more than you do about this particular phase of software engineering. In fact, since we try to assign

projects to those who are best qualified for them, it is likely that the reviewer knows less about this particular speciality than you do. *Regardless of the specialized knowledge or lack of it, the reviewer can still point out flaws in technique and mistakes in work.* It's a safe bet that football coaches aren't as good at quarterbacking as the quarterbacks, but the combination of their experience and outside viewpoint makes their advice very valuable indeed.

It is important to view software engineering, at least the coding aspects, as a science rather than a religion. In religion, you're convinced you're right and the goal is to convince your opponent that he is wrong. In science, you *want* to be right and you examine eagerly your colleges arguments in the expectation that you will learn something new. In religion you *know* that you're right, in science you want to *find out* what is right. Top-level architecture issues and lowest- level lexical style issues can be religious. The top level architecture is not at issue during a code review and the low-level lexical style is arbitrarily declared in the Microsoft Standard. Otherwise, we're all working towards the same goals of correctness, structure, size and speed. By freely discussing the pros and cons of the alternatives, we should nearly always reach a consensus on the correct decision.

As mentioned, its easy to take a change to your program as criticism of yourself as a programmer... especially when you are clearly wrong. Although it is difficult to argue because you *are* clearly wrong, such a viewpoint makes one defensive and less receptive to the remaining issues which may not be so black and white. Its important to remember that humans are biological creatures and have a certain innate error rate.

It helps to consider yourself a "coding machine" that you have designed, built, and are continually upgrading. If you had such a machine, you would measure the quality of the items the machine is creating, both by direct measurement and by reading the letters received by the customer service department (the code review). When a defective unit is found, it is not taken as a personal criticism. Instead, the defect is dispassionately analyzed:

> Was there bad raw material (Poor specification)
> Was the machine built incorrectly (Bad design)
> Was the process defective (Incorrect or inadequate tools)
> Was there an error in quality control (Poor testing)

After this analysis is complete, two steps would be taken:

> The source of the error would be adjusted to reduce further errors
> Tests would be run to make sure the problem was corrected

You wouldn't take returned or defective units as personal criticism, you'd take pride in paying careful attention to such units so that you can reduce their number to the absolute minimum. This is exactly how you should treat yourself and your own error rate. No human will ever have a zero error rate, but you should treat each error as an opportunity to trim your rate even lower. Its OK to make errors, its not OK to be negligent or to make careless errors. If you do your work carefully, 'criticism' of your work is not criticism of yourself; it is feedback to help you become even better than you are.

How to Maximize the Benefit of a Code Review

One of the nicest things about a code review is that it can improve code quality before it takes place. Since you know what kind of code the reviewer is going to demand, you'll naturally write it that way in the first place. You'll discover that structure and

documentation techniques designed to help others work with your code also help *you* to work with it, as well. Many times, when working on my code, I'm in the process of structuring and commenting it to demonstrate to a future reviewer that it covers all the bases when I discover that it doesn't cover them after all! Likewise, the strong documentation and design standards are a great help in desk checking, debugging, and later using the code for further work.

My favorite technique is to envision a hypothetical reviewer who is somewhat obsessive. I challenge myself to write code that such a reviewer can find nothing to complain about. In areas where I think I've done a just good enough job, I go back and touch it up so there will be absolutely no grounds for any adjustment. At first this process takes conscious effort and extra time. However, it soon becomes habitual. The large benefits which result far outweigh the rather small initial overhead of producing clean code.

As I emphasized above, it is important that you take the review comments as feedback, not as criticism. The reviewer's job is semi-mechanical; The code is double-checked against the 'design rules' (coding standards) and the understandability is evaluated by the acid test of trying to understand what the code is doing. The report to you will be a simple, non-critical measurement on how well your program does or doesn't meet these goals.

Sometimes a reviewer's comments are inappropriate because of failure to understand some aspect of your design and therefore suggested changes are inappropriate. *Such a case is still valuable feedback.* If the reviewer misunderstood your code then it was not structured and/or documented clearly enough. Its like a beta-tester for your new "foobar Model X" circuit board plugging it in backwards and blowing it up. You don't criticise the tester, you say "We are really lucky that problem turned up in beta-test and not after production!"

Hints for the Reviewer

Be Objective

Read the above discussions for the author, and invert. Remember that until the author becomes accustomed to the review process, there will be a tendency to take comments personally. Consider the author's state of mind as the review is read and word the review to allow the author to accept your suggestions with a minimum of bruising. He'll be more receptive and consequently more productive.

When Uncertain, Ask

A common reason for review comments is that you don't understand something, or a code segment appears superfluous but you're not sure. In such cases don't say "this is wrong, it should be like this..." but *ask* the author what the situation is and perhaps discuss a couple of alternatives. If one of your possible interpretations is the right one, you'll save a 'turn around' on the review. For example (discussing a macro *paddr*):

> What is the purpose of 'paddr'? Can't you just have the assembler/linker give you the 'segment #' of the headers and avoid the arithmetic? If not that, how about just computing the values for the free and busy chain and storing those somewhere?

As I emphasized above, the review process should be seen as a technical discourse about a scientific subject. When you want to make a point but you're not 100% sure

that the point is correct then use a phrase such as "I argue that..." or "I believe that...". These phrases are important because they communicate the distinction between your belief that something is so and your statement that something is so (and can presumably prove it.)

Suggestions vs Demands

Pay attention to how you phrase your comments, be they 'suggestions' or 'demands'. A suggestion is better received than a demand. When the degree of sin is small, a suggestion is definitely called for:

```
Routine gethandle: isn't this a misnomer?  Isn't
the function of this routine to get an address?
I'd call it CHA (Convert Handle to Address)
```

In this case a new name was suggested because the key item was the fact that the current name was wrong. The exact form that the new name will have is less important.

The reviewer must adjust to the author. You must be careful to not go overboard in making 'general suggestions' and risk having the author not appreciate or understand the changes that have to be made. When you're not sure that the author will get the hint, spell it out a bit more clearly. More on this later.

When you make a correction, discuss why it is better. This is important because the author is supposed to be using the review feedback to improve skills. Since we're talking science rather than religion, you should back up your suggestion with the underlying justification so that, in case the reviewer is wrong, the author can counter-argue.

When commenting on a religious issue, there may be no 'scientific reason'. In that case, just "quote the little red book". Examples:

```
Note warnings about not removing block from free
list, re-entrancy, etc.  Stuff like this is
necessary because slipups in reentrancy will give
us bugs that it will take years of work to find.
That means it is worth the investment of 'overkill'
with regards to documentation, warnings, etc.

Comment sub-banners should have their ";" at the
left and be surrounded by white space. Major comment
subbanners have a ";*" in col 1 and 2 blanks before
them.  So sayeth the book.

I convoluted the loop at galloc1:; since 'taken'
jumps are very expensive this is actually faster
in both the found and not found cases and is
smaller as well.
```

Note that the last example was addressed to a beginner at 8086 assembly language coding and therefore explained something that might not have been obvious: the timing difference between taken and not-taken jumps. If the author were known to be an experienced 8086 hacker such detail might be offensive. The comment would then have read:

```
I convoluted the loop at galloc1: for speed.
```

/HU "Repeat Some"

Don't repeat an identical comment over and over as it applys to different lines of code; it gives the impression of "counting coup". On the other hand, don't just say it once and assume that the author will think to make all the corrections. Repeat the comment a couple of times, perhaps, then make reference to the ones you're omitting:

```
Again, signed compares should be unsigned compares.
Check all jumps, I have likely missed some.
```

Show Improved Examples

Its often more constructive to *show* what you suggest, rather than just talking about it. For example:

```
Note that instead of comments like:
       mov     cx, es     ; save the pointer
You can say
       mov     cx, es     ; (cx) = segment of block
which "refreshes" the reader's memory more.  Its
also easy to look back and see what the contents
of (cx) are.
```

Often its convenient just to make the proposed edits to the source under review, then refer to what and why in the review memo:

```
I convoluted the "space big enough" jumps to reduce
the number of instructions in the loop cause it's
inner-most.  Also, I'd try making the sizes of the
special head and tail nodes FFFF so that I could
shorten the loop still further and just say
"is this big enough" and if the answer is yes, then
test to see if we've actually run to the end.
```

In the above example the reviewer 'demo-ed' one proposed change and just described another which involved more sweeping changes.

Suggest, Rather than Command

Phrasing your comments as suggestions helps defuse ego-involvement. In return, the author must not ignore such suggestions but give them full consideration. The author should either take the suggestion or counter-argue the point with the reviewer. Don't just let the suggestion 'drop on the floor'; close the loop by informing the reviewer which suggestions you are passing up, and why.

Suggestions that involve alternative ways of writing something are best presented as a synopsis of a general technique, followed by pointers to instances in the code where this technique might be applied. Since it's been presented as a general technique, the author will be much more likely to use it again in the future.

If your suggestion is more of a specific nature, try to word it as a question- "Have you considered foo?" Elaborate a little on the tradeoffs that you see.

Don't Over-Specify

When you point out a problem whose proper solution is clearly within the capabilities of the author, don't spell it out in insulting detail. Do be sure to discuss it fully, but when you can, leave the details to the programmer. For example:

> If a guy keeps trying to shrink his segment by something less than minsize it will never get smaller. We need to keep both the TRUE size and the REQUESTED size of the block and reduce the REQUESTED size. When that becomes an allocation unit smaller than we can calve off a free block. We also need the requested size for handling the 'sizeof' call.

Don't Under-Specify

Worse than over-specification is under-specification. You must make your comments sufficiently clear and precise so that the author can clearly understand you. Don't let your suggestions flunk their own review due to vagueness or incompleteness. When you've just sketched out a solution, make sure you emphasize the inadequacy of the sketch. For example:

> As I understand it, we're looking to see if the block following a block is free. Why not just look? We know its header address. I changed the code in grealloc to do just this. One bug I've introduced, though, is that we'll have to 'terminate' memory with a special block marked 0 length and in use. Don't need to put it on any chain, but it has to be there in case we try to extend a block which abuts high memory.

Naturally, the level of detail necessary depends upon the author's skill and familiarity with the techniques you suggest. Better too much detail than too little; the author should understand that you're just making sure and not take offense.

Give Positive Feedback

Finally, after pages of suggestions and discussion of techniques to correct problems, don't forget to put in a little positive feedback. Its not the reviewer's job to embed comments in flattery - the purpose of the review is to point out areas that could be touched up. The author understands this and is not expecting a lot of 'stroking'. However, when you see something has been done well, spend a few bytes of email and say so.

As I alluded to above, the sincerest form of compliment from a reviewer is a a short review memo. If you receive only a few comments and those comments are minor then the reviewer is telling you that he doesn't see how it could be done better, and thats a true compliment.

Appendix

This appendix consists of excerpts from a real-world program which adheres closely to these coding and design specs. Look this over and see how well you can understand what these routines do and how they do it. Imagine yourself being given the task to make some change or extension to this code: how quickly could you find the area(s) to be modified, how easily could you determine what the changes should be, and how likely is it that your changes might inadvertently break something else?

Some specific notes:

Page 1 & 2

Unfortunately MASM requires a lot of obscure declarations at the beginning of the program. Note that effort has been made to split this stuff up into managable hunks. Also note the initial banner which makes it clear, literally at first glance, what these routines do, and why.

Page 3

Note the use of "illustrations" to make a record's structure clear at a glance. Note that the programmer has gone to some effort to keep the communications density high while retaining clarity.

Page 4

The "NOTE:" comments point out special assumptions, requirements, or other items of importance that may not be directly obvious.

Page 5

The "NOTE:" comment here anticipates a potential problem for future maintainers and spells out the hidden peril. Again on page 7 a NOTE describes a situation which, although perferectly legitimate, might cause problems for future maintainers who might not pick up on the possibility of trailing garbage after the 00 byte.

Page 9

The author writes down his "proof" that this routine will not orphan a "name" record... to ensure that future changes here will not cause such orphans to become possible and to provide a signpost for reviewers looking for potential problems.

Page 15

Chk_Block must be fast; the author explicitly notes this requirement.

Page 20

The share tables are very complex and prone to human error as well as future changes. The author has gone to a lot of work to try to make them understandable so that he and others can verify the correctness of the tables and have a ghost of a chance of correctly changing them in the future. In this case the density and complexity of the table called for exceptional documentation effort.

Page 22

The code at "cuc8" may jump-up to cuc20. The code structure is still clean and understandable, so the author elected to note the "gotcha" rather than restructure the code to remove it.

Page 27

Note the WARNING:. Hidden among the algorithems is a meta- algorithem which terminates the "can't find space; garbage collect; look again for space"

loop. Its important that the author prove to himself and his reviewer that this loop completes; he also prevents a future maintainer from accidentally introducing such a bug (which would probably get through testing and out into the field).

```
TITLE   Sharer - MultiProcess File Share and Lock Support
NAME    Sharer

;**     Sharer
;
;       SCCSID = @(#)gshare.asm 1.1 84/06/21
;
;       Sharer is a component of DOS 3; it maintains the Master File
;       Table (MFT) and uses it to keep track of all open files and
;       all record locks to those files.
;
;       The sharer is called to check the validity of and register/
;       de-register file opens and record locks.
;
;       Modification history:
;
;           GL 09 Sept 1983 - Created
;           MZ 18 Sept 1983
;               - modified for DOS3 installability

.xlist
.xcref
        include stdsw.inc
        INCLUDE fcbsym.inc
        INCLUDE sft.inc
.cref
.list
.sall


Installed       = FALSE                 ; for installed version

;       if we are installed, define the base code segment of the sharer first

IF Installed
Share   SEGMENT PARA PUBLIC 'SHARE'
Share   ENDS
ENDIF

;       include the rest of the segment definitions for normal MSDOS

IF NOT Installed
        include dosseg.inc
ELSE
  DATA  SEGMENT WORD PUBLIC 'DATA'
  DATA  ENDS

  CODE  SEGMENT BYTE PUBLIC 'CODE'
  CODE  ENDS
```



```
  LAST  SEGMENT BYTE PUBLIC 'LAST'
  LAST  ENDS

  DOSGROUP    GROUP   START,CODE,CONSTANTS,DATA,LAST
ENDIF

;       If we are installed, define all of the data.

IF Installed

START   SEGMENT PARA PUBLIC 'START'
          DB 3 DUP (?)
START   ENDS
        include MSDATA.ASM

ENDIF

;       if we are not installed, then the code here is just part of the normal
;       MSDOS code segment otherwise, define our own code segment

IF NOT INSTALLED
CODE    SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  SS:TaskArea,CS:DOSGROUP
ELSE
SHARE   SEGMENT PARA PUBLIC 'SHARE'
        ASSUME  SS:TaskArea,CS:SHARE
ENDIF
```

```
BREAK   <MFT Definitions>

;**     MSDOS MFT definitions

        The Master File Table (MFT) associates the cannonicalized pathnames,
        lock records and SFTs for all files open on this machine.

        The MFT implementation employs a single memory buffer which is used
        from both ends.  This gives the effect (at least until they run into
        each other) of two independent buffers.

        MFT buffer
        ==========

        The MFT buffer contains MFT name records and free space.  It uses a
        classic heap architecture:  freed name records are marked free and
        conglomerated with any adjacent free space.  When one is to create a
        name entry the free list is searched first-fit.  The list of name and
        free records is always terminated by a single END record.

        LOCK buffer
        ===========

        The lock buffer contains fixed format records containing record
        locking information.  Since they are fixed format the space is handled
        as a series of chains:  one for each MFT name record and one for the
        free list.

        Space allocation
        ================

        The MFT is managed as a heap.  Empty blocks are allocated on a
        first-fit basis.  If there is no single large enough empty block the
        list is garbage collected.

        MFT name records:

                                        Unique file id
                                      /--------------\
           8     16    16     32     16      16       16       8         n
        +------+-----+-----+------+------+--------+--------+--------+----------+
        | FLAG | LEN | SUM | SPTR | LPTR | DIRSEC | DIRPOS | DRVNUM | string   |
        +------+-----+-----+------+------+--------+--------+--------+----------+

                  FLAG =   record type flag
                  LEN =  total byte length of record.
                  SUM  =   sum of idenitifier fields.  Used to speed
                           searches
```



```
                  SPTR =   pointer to first SFT in SFT chain
                  LPTR =   pointer to first record in lock chain segment
                           is MFT segment
                  DIRSEC = The directory sector number of the file.
                  DIRPOS = The file's position in the directory sector.
                  DRVNUM = The file's drive number (A=0).
                  string = name string, zero-byte terminated.  There
                           may be garbage bytes following the 00 byte;
                           these are counted in the LEN field.

        NOTE 1 : The fields DIRSEC, DIRPOS, and DRVNUM make up a unique id
        for the file.

        NOTE 2 : The MFT records for devices set DIRPOS:DIRSEC to be a ptr
        to their device driver and DRVNUM is set to FFh.


    MFT free records

           8      16
        |------|-----|---------------------------------------|
        | FLAG | LEN |          free                         |
        ------------------------------------------------------

                  FLAG = record type flag
                  LEN  = total byte length of record.


    MFT END records

           8
        |------|
        | FLAG |
        --------

                  FLAG = record type flag

        NOTE 1 : The last 3 fields of the record make up a unique identifier
        for the file.

        NOTE 2 : The MFT records for devices set DIRPOS:DIRSEC to be a ptr
        to their device driver and DRVNUM is set to FFh.


;**     MFT definitions
;*
;*      NOTE:  the flag and length fields are identical for all record types
;*             (except the END type has no length) This must remain so as
```

```
;*              some code depends upon it.
;*
;*      NOTE:  Many routines check for "n-1" of the N flag values and if no
;*              match is found assume the flag value must be the remaining
;*              possibility.  If you add or remove flag values you must check
;*              all references to mft_flag.

MFT_entry       STRUC

mft_flag        DB      ?               ; flag/len field
mft_len         DW      ?
mft_sum         DW      ?               ; string sum word
mft_sptr        DD      ?               ; SFT pointer
mft_lptr        DW      ?               ; LCK pointer
mft_dirsec      DW      ?               ; directory sector
mft_dirpos      DW      ?               ; position in directory sector
mft_drvnum      DB      ?               ; drive number
mft_name        DB      ?               ; offset to start of name

MFT_entry       ENDS

 MFLG_NAM EQU   1                               ; min value for name record
 MFLG_FRE EQU   0                               ; free record
 MFLG_END EQU   -1                              ; end record


;**     Record Lock Record (RLR):
;
;                  16      32      32      32
;                 |-------|-------|-------|-------|
;                 | NEXT  |  FBA  |  LBA  | SPTR  |
;                 |       | lo hi | lo hi |       |
;                 ---------|-------|-------|--------
;
;                        CHAIN  = pointer to next RLR.  0 if end
;                        FBA    = offset of 1st  byte of locked region
;                        LBA    = offset of last byte of locked region
;                        SFTPTR = pointer to SFT lock was issued on

RLR_entry       STRUC

rlr_next        DW      ?               ; chain to next RLR, 0 if end
rlr_fba         DW      ?               ; first byte addr (offset) of reigion
                DW      ?
rlr_lba         DW      ?               ; last byte addr of region
                DW      ?
rlr_sptr        DD      ?               ; SFT pointer

RLR_entry       ENDS
```

```
BREAK   <MFT and Lock Record Data Area>

;**     First MFT record
;
;       Note that the name field can have garbage after the trailing
;       00 byte.  This is because the field might be too long, but
;       not long enough (at least 16 extra bytes) to fragment.
;       in this case we copy the length of the string area, not
;       the length of the string and thus may copy tailing garbage.

MFT:
        DB      0               ; free
        DW      490             ; 490 bytes long
        DB      487 DUP(0)      ; leave rest of record
MEND    DB      -1              ; END record

lck1    DW      0               ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck2    DW      lck1            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck3    DW      lck2            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck4    DW      lck3            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck5    DW      lck4            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck6    DW      lck5            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck7    DW      lck6            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck8    DW      lck7            ; link
        DB      SIZE RLR_entry-2 DUP(0)


Frelock DW      lck8            ; Ptr to lock free list
```



```
BREAK <Sharer - MultiProcess File Sharer>

;**     MSDOS MFT Functions
;
;       The Master File Table (MFT) associates the cannonicalized pathnames,
;       lock records and SFTs for all files open on this machine.
;
;       These functions are supplied to maintain the MFT and extract
;       information from it.  All MFT access should be via these routines so
;       that the MFT structure can remain flexible.
;


BREAK <Mft_enter - Make an MFT entry and check access>

;**     mft_enter - make an entry in the MFT
;
;       mft-enter is called to make an entry in the MFT.
;
;       mft_enter checks for a file sharing conflict:
;               No conflict:
;                   A new MFT entry is created, or the existing one updated,
;                   as appropriate.
;               Conflicts:
;                   The existing MFT is left alone.  Note that if we had to
;                   create a new MFT there cannot be, by definition, sharing
;                   conflicts.
;
;       ENTRY   ThisSFT points to an SFT structure.  The sf_mode field
;                   contains the desired sharing mode.
;               Path_name contains a long pointer to the full pathname for
;                   the file
;               Uid = 16-bit user id of issuer
;               Pid = 16-bit process id of issuer
;               (CS) = DOSGroup             BUGBUG - installed
;               [OPEN DEVID] = set appropriately
;               If FILE
;                   [CURBUF+2]:BX  points to start of directory entry
;                   [THISDRV]  set to drive number of file.
;               If device
;                   [DEVPT] = has DWORD pointer to device driver
;
;       EXIT    'C' clear if no error'
;               'C' set if error
;                 (ax) = error code
;       USES    ALL but DS

Procedure   mft_enter,NEAR
```

```
    ASSUME  ES:NOTHING
        push    ds                      ; preserve (DS)

;       find or make a name record

        lds     si,Path_Name            ; (DS:SI) = FBA of file name
        DEBUG   1,8,<MFT_ENTER 1:8 - e$x:$x Name=$s\n>,<DS, SI, DS, SI>
        mov     al,1                    ; allow creation of MFT entry
        push    es
    ASSUME  DS:NOTHING
        call    FNM                     ; find or create name in MFT
        pop     es
        mov     ax,error_not_enough_memory
        LJC     ent9                    ; not enough space
        DEBUG   1,8,<Survived FNM ($x)>,<bx>

;       add the SFT to the chain
;
;       (bx) = fwa name record

        lds     si,ThisSFT
        call    ASC                     ; try to add to chain

;       As noted above, we don't have to worry about an "empty" name record
;       being left if ASC refuses to add the SFT - ASC cannot refuse if we had
;       just created the MFT...

;       return.
;
;       'C' and (Ax) setup appropriately

ent9:   pushf
        xor     cx,cx
        popf
        jnc     ent10                   ; (CX) = 0 if OK
        not     cx                      ; (CX) = -1 if error
ent10:  DEBUG   1,8,<MFT_ENTRY: carry = $x  ax = $x\n>,<cx,ax>
        pop     ds
        return

EndProc     mft_enter
```



```
BREAK <MftClose - Close out an MFT for given SFT>

;**     MFTclose - Close an SFT/MFT Entry
;
;       MFTclose(SFT)
;
;       MFTclose removes the SFT entry from the MFT structure.  If this was
;       the last SFT for the particular file the file's entry is also removed
;       from the MFT structure.
;
;       Note that the SFT refcount has not yet been decremented by our caller.
;       A refcount of 1 means that the SFT is going idle
;
;       ENTRY   (ES:DI) points to an SFT structure
;               (DS) = (CS) = DOSGroup
;       EXIT    NONE
;       USES    ALL but DS, ES:DI

Procedure   MFTclose,NEAR
    ASSUME  ES:NOTHING
        mov     ax,es:[di].sf_MFT
        or      ax,ax
        jz      mcl10                   ; No entry for it, ignore ('C' clear)
        push    ds                      ; preserve regs
        push    es
        push    di
        call    CCL                     ; clear SFT locks
    ASSUME  DS:NOTHING
```

```
BREAK <MFT_get - get an entry from the MFT>

        MFT_get - get an entry from the MFT

;**
;       MFT_get is used to return information from the MFT.  System utilities
;       use this capability to produce status displays.
;
;       MFT_get first locates the (BX)'th file in the list (no particular
;               ordering is promosed).  It returns that name and the
;               UID of the (CX)'th SFT on that file and the number
;               of locks on that file via that SFT.
;
;       ENTRY   (BX) = zero-based file index
;               (CX) = zero-based SFT index
;               (ES:DI) point to buffer for file name
;               (SS) = DOSGroup
;       EXIT    'C' clear if no error
;                 ES:DI buffer is filled in with BX'th file name
;                 (BX) = user id of SFT
;                 (CX) = # of locks via SFT
;               'C' set if error
;                 (ax) = error code
;                         ('error_no_more_files' if either index is out
;                          of range)
;

Procedure   MFT_get,NEAR

    ASSUME  DS:NOTHING,ES:NOTHING

        xchg    bx,cx                   ; (cx) = file index
        mov     ax,CS
        mov     ds,ax

        mov     si,OFFSET DOSGROUP:MFT  ; (ds:si) = fwa of OFFSET MFT

        scan forward until next name
;
mget1:  test    [si].mft_flag,-1
        jz      mget3                   ; is free space
        js      mget7                   ; is END

        have another name.  see if this satisfies caller
;
        jcxz    mget4                   ; caller is happy
        dec     cx
mget3:  add     si,[si].mft_len         ; skip name record
        JMP     SHORT mget1

        we've located the file name.
;
```



```
;
;       (bx) = SFT index
;       (DS:SI) = MFT entry
;       (ES:DI) = address of caller's buffer

mget4:  push    di
        push    si
        add     si,mft_name             ; save table offset
mget5:  lodsb
        stosb
        and     al,al                   ; copy name into caller's buffer
        jnz     mget5
        pop     si
        xchg    bx,cx                   ; (DS:SI) = name record address
        lds     si,[si].mft_sptr        ; (cx) = SFT chain count
mget6:  jcxz    mget8
        dec     cx                      ; have reached the SFT we wanted
        lds     si,[si].sf_chain
        mov     ax,ds
        or      ax,si
        jnz     mget6
        pop     di                      ; follow chain some more
                                        ; (es:di) = buffer address
;**     The file or SFT index was too large - return w/ error

mget7:  mov     ax,error_no_more_files
        stc
        ret


;**     We've got the SFT he wants.  Lets count the locks
;
;       (es:TOS) = buffer address
;       (DS:SI) = address of SFT

mget8:  mov     dx,ds
        mov     di,si
        mov     si,[si].sf_MFT          ; (dx:DI) = SFT address
        mov     ax,CS
        mov     ds,ax
        mov     si,[si].mft_lptr        ; (DS:SI) = Lock record address
        sub     cx,cx                   ; clear counter

mget9:  or      si,si
        jz      mget11                  ; no more
        cmp     di,WORD PTR [si].rlr_sptr
        jnz     mget10
        cmp     dx,WORD PTR [si].rlr_sptr+2
        jnz     mget10
        inc     cx
```

```
mget10: mov     si,[si].rlr_next
        JMP     SHORT mget9
;
;       Done counting locks.  return the info
;
;       (cx) = count of locks
;       (es:TOS) = buffer address
;       (dx:di) = SFT address

mget11: mov     ds,dx
        mov     bx,[di].sf_UID          ; (bx) = UID
        pop     di
        clc
        ret

EndProc MFT_get
```



```
BREAK <CUC - check usage conflicts>

;**     Use conflict table
;
;       Algorithm:
;
;               for new and old mode, compute index of (SH*3)+ACC
;               shift left table[new_index] by old_index+1;
;                       'C' set if FAIL
;
;       The bit in the old_index position indicates the success or failure.  0
;       => allow access, 1 => fail access
;
                                                ; Compat      Read
CUCA:   DW      0df7dh                          ; Compat      Write
        DW      0ffffh                          ; Compat      Read/Write
        DW      0ffffh                          ; Deny R/W    Read
        DW      0ffffh                          ; Deny R/W    Write
        DW      0ffffh                          ; Deny R/W    Read/Write
        DW      0ffffh                          ; Deny W      Read
        DW      0df7dh                          ; Deny W      Write
        DW      0dbffh                          ; Deny W      Read/Write
        DW      0dfffh                          ; Deny R      Read
        DW      0beffh                          ; Deny R      Write
        DW      0b7ffh                          ; Deny R      Read/Write
        DW      0bfffh                          ; Deny None Read
        DW      01c7dh                          ; Deny None Write
        DW      003ffh                          ; Deny None Read/Write
        DW      01ffffh

                                                 4443 3322 2111 000
                                                 DDDD DDDD DDDD CCCx
;                                               /   R RR   RRR
;       Deny/Compat                            /       WW WWWW
;       DenyRead                1st Access =<
;       DenyWrite                              \  R RR  RR  RR R R R
;       AccessRead                              \ WW W W WW  WW  WW
;       AccessWrite                               1111 1111 1111 1111
;       x                                         1101 1111 0111 1101   df7d
;       C  R                                      1111 1111 1111 1111   ffff
;       C   W                                     1111 1111 1111 1111   ffff
;       C  RW
;                                                 111: 1111 1111 1111   ffff
;       DRWR                                      111: 1111 1111 1111   ffff
;       DRW W                                     1111 1111 1111 1111   ffff
;       DRWRW                                     1101 1111 0111 1101   df7d
;       D WR
;                                                 1101 1011 1111 1111   dbff
;       D W W                                     1101 1111 1111 1111   dfff
;       D WRW                                     1011 1110 1111 1111   beff
;       DR R                                      1011 0111 1111 1111   b7ff
;       DR  W
```

```
;   DR RW                                   1011 1111 1111 1111  bfff
;   D  R                                    0001 1100 0111 1101  1c7d
;   D   W                                   0000 0011 1111 1111  03ff
;   D  RW                                   0001 1111 1111 1111  1fff
;
;   In order to allow the greatest number of accesses, compatability read mode
;   is treated as deny-write read.  The other compatability modes are treated
;   as deny-both.


;**     CUC - check usage conflicts
;
;       CUC is called to see if a would-be open would generate a share
;       conflict with an existing open.
;       See CUCA for the algorithm and table format.
;
;       ENTRY   (BX) = FBA MFT name record
;               (DS:SI) = SFT address
;       EXIT    'C' clear if OK
;               'C' set if conflict
;                 (ax) = error code
;       USES    ALL but arguments (BX, DS:SI)

CUC:    mov     ax,ds
        mov     es,ax
        mov     di,si                   ; (es:di) = FBA SFT record
        mov     al,BYTE PTR [si].sf_mode; (al) = mode byte
        mov     ch,al
        and     ch,sharing_mask         ; (ch) = new guy share
        jz      cuc0                    ; new guy is compatability mode
        mov     ch,sharing_mask
cuc0:   call    csi                     ; compute share index
        add     ax,ax                   ; *2 for word index
        xchg    ax,si                   ; (si) = share table index
        mov     dx,WORD PTR CUCA[si]    ; (dx) = share mask
        mov     ax,cs
        mov     ds,ax                   ; (ds:bx) = FBA MFT record
        lds     si,[bx].sft_sptr        ; (si) = first SFT guy

;       ready to do access compares.
;
;       (ds:si) = address of next SFT
;       (es:di) = address of new  SFT
;       (dx) = share word from CUCA
;       (bx) = MFT offset
;       (ch) = 0 if new SFT is compatibilty mode, else sharing_mask

cuc1:   mov     ax,ds
```



```
        or      ax,si
        LJZ     cuc9                    ; at end of chain, no problems
        mov     al,BYTE PTR [si].sf_mode; (al) = mode byte
        DEBUG   1,8,< chk $b/$x mask $x -- >,<ax, cx, dx>

cuc2:   call    csi                     ; compute the share index
        inc     ax
        inc     ax
        xchg    al,cl                   ; (cl) = shift count
        mov     ax,dx
        sar     ax,cl                   ; select the bit
        DEBUG   1,8,< $x~$x >,<ax,dx>
        jc      cuc8                    ; a conflict! (probably)

;       we may come back here from "cuc8" if the conflict is a false alarm...

cuc20:  lds     si,[si].sf_chain
        JMP     cuc1                    ; chain to next SFT and try again

;       Have a share conflict

cuc8:   cmp     open_devid,10000000b    ; If the file is a device that has been
        jz      cuccon                  ;  opened in deny none mode and the
        mov     al,byte ptr [si].sf_mode;  current open is in compatibility
        test    al,01000000b            ;  mode, allow the open.
        jz      cuccon                  ; Otherwise, report the conflict.
        mov     al, byte ptr es:[di].sf_mode
        test    al,11110000b
        jz      cuc20

cuccon: mov     ax,error_sharing_violation  ; assume share conflict
        DEBUG   1,8,<CUC - SHARE VIOLATION>,<>
        stc

;       done with compare.  Restore regs and return
;
;       'C' set as appropriate
;       (es:di) = new SFT address
;       (ax) set as appropriate
;       (bx) = MFT offset

cuc9:   mov     cx,es
        mov     ds,cx
        mov     si,di
        ret


BREAK <csi - compute share index>
```

```
;**     csi - compute share index
;
;       csi turns a mode byte into an index from 0 to 14:
;
;               (share index)*3 + (access index)
;
;       ENTRY   (al) = mode byte
;       EXIT    (ax) = index
;       USES    AX, CL

csi:    mov     ah,al
        DEBUG   1,8,<CSI mode $b gives >,<ax>
        and     ah,access_mask          ; (ah) = access bits
        and     al,sharing_mask         ; (al) = share bites
        ERRNZ   sharing_mask-0f0H
        shr     al,1
        shr     al,1
        shr     al,1
        mov     cl,al                   ; (cl) = SHVAL*2
        shr     al,1
        add     al,cl                   ; (al) = SHVAL*3
        add     al,ah                   ; (al) = SH*3 + ACC
        sub     ah,ah
        DEBUG   1,8,<.$d.  >,<ax>
        ret
```



```
BREAK <FNM - Find name in MFT>

;**     FNM - Find name in MFT
;
;       FNM searches the MFT for a name record.
;
;       ENTRY   (DS:SI) = pointer to name string (.asciz)
;               (al) = 1 to create record if non exists
;                    = 0 otherwise
;       EXIT    'C' clear if found or created
;                 (DS:BX) = address of MFT name record
;               'C' set if error
;                 If not to create, item not found
;                   (DS:SI) unchanged
;                 If to create, an out of space
;                   (ax) = error code
;       USES    ALL

FNM:    push    ds                      ; save string address
        push    si
        xchg    bh,al                   ; (bh) = create flag

;       run down through string counting and summing

        sub     dx,dx                   ; (dx) = byte count
        sub     bl,bl                   ; (bl) = sum

fnm1:   lodsb                           ; (al) = next char
        add     bl,al
        inc     dx
        and     al,al
        jnz     fnm1                    ; terminate after null char

;       Info computed.  Start searching name list
;
;       (bh) = create flag
;       (bl) = sum byte
;       (dx) = byte count
;       (TOS+2:TOS) = name string address

        mov     ax,cs
;       mov     ax,SEG CODE
        mov     ds,ax
        mov     si,OFFSET MFT
fnm2:   test    [si].mft_flag,0FFh
        js      fnm10                   ; at end - name not found
        jz      fnm4                    ; is free, just skip it
        cmp     bl,[si].mft_sum         ; do sums compare?
        jz      fnm5                    ; its a match - look further
```

```
fnm4:   add     si,[si].mft_len         ; not a match... skip it
        JMP     SHORT fnm2

;       name checksums match - compare the actual strings
;
;       (dx) = length
;       (DS:SI) = MFT address
;       (bh) = create flag
;       (bl) = sum byte
;       (dx) = byte count
;       (TOS+2:TOS) = name string address

fnm5:   mov     cx,dx           ; (cx) = length to match
        pop     di
        pop     es              ; (di:es) = fba given name
        push    es
        push    di
        push    si              ; save MFT offset
        add     si,mft_name     ; (ds:si) = fwa string in record
        repz    cmpsb
        pop     si              ; (ds:si) = fwa name record
        jnz     fnm4            ; not a match

;       Yes, we've found it.  Return the info
;
;       (TOS+2:TOS) = name string address

        pop     ax              ; discard unneeded stack stuff
        pop     ax
        mov     bx,si           ; (ds:bx) = fwa name record
        clc
        ret


;**
;**     Its not in the list - lets find a free spot and put it there
;**

;       (bh) = create flag
;       (bl) = sum byte
;       (dx) = string length
;       (TOS+2:TOS) = ASCIZ string address
;       (ds) = SEG CODE

fnm10:  and     bh,bh
        jnz     fnm10$5         ; yes, insert it
        pop     si
        pop     ds              ; no insert, its a "not found"
        stc
        mov     ax,error_path_not_found
        ret
```



```
fnm10$5:add     dx,mft_name             ; (dx) = minimum space needed
        mov     si,OFFSET MFT
fnm11:  test    [si].mft_flag,OFFh
        js      fnm20                   ; at END, an out of space
        jz      fnm12                   ; is a free record
        add     si,[si].mft_len         ; skip name record
        JMP     SHORT fnm11

fnm12:  mov     ax,[si].mft_len         ; Have free record, (ax) = total length
        cmp     ax,dx
        jnc     fnm13                   ; big enough
        add     si,ax
        JMP     SHORT fnm11             ; not large enough - move on

;       OK, we have a record which is big enough.  If its large enough
;       to hold another name record of 6 characters than we'll split
;       the block, else we'll just use the whole thing
;
;       (ax) = size of free record
;       (dx) = size needed
;       (ds:si) = address of free record
;       (bl) = sum byte
;       (TOS+2:TOS) = name string address

fnm13:  sub     ax,dx           ; (ax) = total size of proposed fragment
        cmp     ax,mft_name+6
        jc      fnm14           ; not big enough to split
        push    bx              ; save sum byte
        mov     bx,dx           ; (bx) = offset to start of new name record
        mov     [bx][si].mft_flag,MFLG_FRE
        mov     [bx][si].mft_len,ax     ; setup tail as free record
        sub     ax,ax           ; don't extend this record
        pop     bx              ; restore sum byte
fnm14:  add     dx,ax           ; (dx) = total length of this record
        mov     [si].mft_len,dx
        mov     [si].mft_sum,bl
        mov     [si].mft_flag,MFLG_NAM
        mov     ax,ds
        mov     es,ax           ; (es) = MFT segment for "stow"
        sub     ax,ax
        mov     di,si
        add     di,mft_sptr
        stosw
        stosw                   ; zero SFT pointer
        ERRNZ   mft_lptr-mft_sptr-4
        stosw                   ; zero LCK pointer

;       we're all setup except for the name.
;       Note that we'll block copy the whole name field, even though
```

```
;       the name may be shorter than that (we may have declined to fragment
;       this memory block)
;
;       (dx) = total length of this record
;       (ds:si) = address of working record
;       (es) = (ds)
;       (TOS+2:TOS) = name string address

        mov     cx,dx
        sub     cx,mft_name     ; compute total size of name area
        ERRNZ   mft_name-mft_lptr-2
        xchg    ax,si           ; save name record offset
        pop     si
        pop     ds
        rep     movsb
        xchg    bx,ax           ; (bx) = name record offset
        mov     ax,es
        mov     ds,ax           ; (DS:BX) = name record offset
        clc
        ret

;**
;**     OUT OF FREE SPACE
;**
;**     This is tough, folks.  Lets trigger a garbage collection
;**     and see if theres enough room.  If there is, we'll hop back
;**     and relook for a free hunk; if there isnt enough space, its
;**     error-city!
;
;       WARNING:  it is important that the garbage collector be told
;               how big a name record hole we're looking for... if the size
;               given GCM is too small we'll loop doing "no space; collect;
;               no space; ...)
;
;       (dx) = total length of desired name record
;       (ds) = SEG CODE
;       (bl) = sum byte
;       (TOS+2:TOS) = name string address

fnm20:
        mov     ax,dx           ; (ax) = size wanted
        sub     dx,mft_name     ; (dx) = string length for reentry at fnm10
        push    dx
        push    bx
        call    GCM             ; garbage collect MFT
        pop     bx
        pop     dx
        jnc     fnm10           ; go back and find that space

;       no space, return w/error
```



```
        pop     ax
        pop     ax              ; clean stack
        mov     ax,error_not_enough_memory
        stc
        ret
```

```
TITLE   Sharer - MultiProcess File Share and Lock Support
NAME    Sharer

;**     Sharer

        SCCSID = @(#)sshare.asm 1.1 84/08/21

        Sharer is a component of DOS 3; it maintains the Master File
        Table (MFT) and uses it to keep track of all open files and
        all record locks to those files.

        The sharer is called to check the validity of and register/
        de-register file opens and record locks.

        Modification history:

            GL 09 Sept 1983 - Created
            MZ 18 Sept 1983
                - modified for DOS3 installability

.xlist
.xcref
        include stdsw.inc
        INCLUDE fcbsym.inc
        INCLUDE sft.inc
.cref
.list
.sall


Installed       = FALSE                 ; for installed version

;       if we are installed, define the base code segment of the sharer first

IF Installed
Share   SEGMENT PARA PUBLIC 'SHARE'
Share   ENDS
ENDIF

;       include the rest of the segment definitions for normal MSDOS

IF NOT Installed
        include dosseg.inc
ELSE
  DATA  SEGMENT WORD PUBLIC 'DATA'
  DATA  ENDS

  CODE  SEGMENT BYTE PUBLIC 'CODE'
  CODE  ENDS
```



```
  LAST  SEGMENT BYTE PUBLIC 'LAST'
  LAST  ENDS

  DOSGROUP    GROUP   START,CODE,CONSTANTS,DATA,LAST
ENDIF

;       If we are installed, define all of the data.

IF Installed

START   SEGMENT PARA PUBLIC 'START'
          DB 3 DUP (?)
START   ENDS
        include MSDATA.ASM

ENDIF

;       if we are not installed, then the code here is just part of the normal
;       MSDOS code segment otherwise, define our own code segment

IF NOT INSTALLED
CODE    SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  SS:TaskArea,CS:DOSGROUP
ELSE
SHARE   SEGMENT PARA PUBLIC 'SHARE'
        ASSUME  SS:TaskArea,CS:SHARE
ENDIF
```

BREAK <MFT Definitions>

;** MSDOS MFT definitions

The Master File Table (MFT) associates the cannonicalized pathnames, lock records and SFTs for all files open on this machine.

The MFT implementation employs a single memory buffer which is used from both ends. This gives the effect (at least until they run into each other) of two independent buffers.

MFT buffer
==========

The MFT buffer contains MFT name records and free space. It uses a classic heap architecture: freed name records are marked free and conglomerated with any adjacent free space. When one is to create a name entry the free list is searched first-fit. The list of name and free records is always terminated by a single END record.

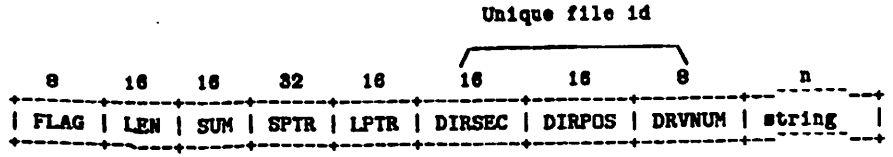
LOCK buffer
===========

The lock buffer contains fixed format records containing record locking information. Since they are fixed format the space is handled as a series of chains: one for each MFT name record and one for the free list.

Space allocation
================

The MFT is managed as a heap. Empty blocks are allocated on a first-fit basis. If there is no single large enough empty block the list is garbage collected.

MFT name records:



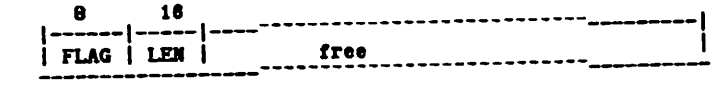
FLAG = record type flag
LEN = total byte length of record.
SUM = sum of idenitifier fields. Used to speed searches

SPTR = pointer to first SFT in SFT chain
LPTR = pointer to first record in lock chain segment is MFT segment
DIRSEC = The directory sector number of the file.
DIRPOS = The file's position in the directory sector.
DRVNUM = The file's drive number (A=0).
string = name string, zero-byte terminated. There may be garbage bytes following the 00 byte; these are counted in the LEN field.

NOTE 1 : The fields DIRSEC, DIRPOS, and DRVNUM make up a unique id for the file.

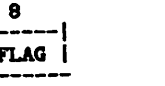
NOTE 2 : The MFT records for devices set DIRPOS:DIRSEC to be a ptr to their device driver and DRVNUN is set to FFh.

MFT free records



FLAG = record type flag
LEN = total byte length of record.

MFT END records



FLAG = record type flag

NOTE 1 : The last 3 fields of the record make up a unique identifier for the file.

NOTE 2 : The MFT records for devices set DIRPOS:DIRSEC to be a ptr to their device driver and DRVNUN is set to FFh.

;** MFT definitions
;*
;* NOTE: the flag and length fields are identical for all record types
;* (except the END type has no length) This must remain so as

```
;*              some code depends upon it.
;*
;*      NOTE:   Many routines check for "n-1" of the N flag values and if no
;*              match is found assume the flag value must be the remaining
;*              possibility.  If you add or remove flag values you must check
;*              all references to mft_flag.

MFT_entry       STRUC

mft_flag        DB      ?               ; flag/len field
mft_len         DW      ?
mft_sum         DW      ?               ; string sum word
mft_sptr        DD      ?               ; SFT pointer
mft_lptr        DW      ?               ; LCK pointer
mft_dirsec      DW      ?               ; directory sector
mft_dirpos      DW      ?               ; position in directory sector
mft_drvnum      DB      ?               ; drive number
mft_name        DB      ?               ; offset to start of name

MFT_entry       ENDS

 MFLG_NAM EQU   1                               ; min value for name record
 MFLG_FRE EQU   0                               ; free record
 MFLG_END EQU   -1                              ; end record


;**     Record Lock Record (RLR):
;
;                 16      32      32      32
;                |-------|-------|-------|-------|
;                | NEXT  |  FBA  |  LBA  |  SPTR |
;                |       | lo hi | lo hi |       |
;                ---------|-------|--------------
;
;                       CHAIN  = pointer to next RLR.  0 if end
;                       FBA    = offset of 1st  byte of locked region
;                       LBA    = offset of last byte of locked region
;                       SFTPTR = pointer to SFT lock was issued on

RLR_entry       STRUC

rlr_next        DW      ?               ; chain to next RLR, 0 if end
rlr_fba         DW      ?               ; first byte addr (offset) of reigion
                DW      ?
rlr_lba         DW      ?               ; last byte addr of region
                DW      ?
rlr_sptr        DD      ?               ; SFT pointer

RLR_entry       ENDS
```

```
BREAK   <MFT and Lock Record Data Area>

;**     First MFT record
;
;       Note that the name field can have garbage after the trailing
;       00 byte.  This is because the field might be too long, but
;       not long enough (at least 16 extra bytes) to fragment.
;       in this case we copy the length of the string area, not
;       the length of the string and thus may copy tailing garbage.

MFT:
        DB      0               ; free
        DW      490             ; 490 bytes long
        DB      487 DUP(0)      ; leave rest of record
MEND    DB      -1              ; END record

lck1    DW      0               ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck2    DW      lck1            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck3    DW      lck2            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck4    DW      lck3            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck5    DW      lck4            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck6    DW      lck5            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck7    DW      lck6            ; link
        DB      SIZE RLR_entry-2 DUP(0)
lck8    DW      lck7            ; link
        DB      SIZE RLR_entry-2 DUP(0)


Frelock DW      lck8            ; Ptr to lock free list
```



```
BREAK <Sharer - MultiProcess File Sharer>

;**     MSDOS MFT Functions
;
;       The Master File Table (MFT) associates the cannonicalized pathnames,
;       lock records and SFTs for all files open on this machine.
;
;       These functions are supplied to maintain the MFT and extract
;       information from it.  All MFT access should be via these routines so
;       that the MFT structure can remain flexible.
;


BREAK <Mft_enter - Make an MFT entry and check access>

;**     mft_enter - make an entry in the MFT
;
;       mft-enter is called to make an entry in the MFT.
;
;       mft_enter checks for a file sharing conflict:
;               No conflict:
;                   A new MFT entry is created, or the existing one updated,
;                   as appropriate.
;               Conflicts:
;                   The existing MFT is left alone.  Note that if we had to
;                   create a new MFT there cannot be, by definition, sharing
;                   conflicts.
;
;       ENTRY   ThisSFT points to an SFT structure.  The sf_mode field
;                   contains the desired sharing mode.
;               Path_name contains a long pointer to the full pathname for
;                   the file
;               Uid = 16-bit user id of issuer
;               Pid = 16-bit process id of issuer
;               (CS) = DOSGroup          BUGBUG - installed
;               [OPEN_DEVID] = set appropriately
;               If FILE
;                   [CURBUF+2]:BX  points to start of directory entry
;                   [THISDRV]  set to drive number of file.
;               If device
;                   [DEVPT] = has DWORD pointer to device driver
;
;       EXIT    'C' clear if no error'
;               'C' set if error
;                 (ax) = error code
;       USES    ALL but DS

Procedure   mft_enter,NEAR
```

```
        ASSUME  ES:NOTHING
        push    ds                      ; preserve (DS)

        find or make a name record

;
        lds     si,Path_Name            ; (DS:SI) = FBA of file name
        DEBUG   1,8,<MFT_ENTER 1:8 - @$x:$x Name=$s\n>,<DS, SI, DS, SI>
        mov     al,1                    ; allow creation of MFT entry
        push    es
        ASSUME  DS:NOTHING
        call    FNM                     ; find or create name in MFT
        pop     es
        mov     ax,error_not_enough_memory
        LJC     ent0                    ; not enough space
        DEBUG   1,8,<Survived FNM ($x)>,<bx>

        add the SFT to the chain

;       (bx) = fwa name record
;
        lds     si,ThisSFT
        call    ASC                     ; try to add to chain

        As noted above, we don't have to worry about an "empty" name record
;       being left if ASC refuses to add the SFT - ASC cannot refuse if we had
;       just created the MFT...
;
        return.

;       'C' and (Ax) setup appropriately
;
ent0:   pushf
        xor     cx,cx
        popf
        jnc     ent10                   ; (CX) = 0 if OK
        not     cx                      ; (CX) = -1 if error
ent10:  DEBUG   1,8,<MFT_ENTRY: carry = $x  ax = $x\n>,<cx,ax>
        pop     ds
        return

EndProc     mft_enter
```



```
BREAK <MftClose - Close out an MFT for given SFT>

;**     MFTclose - Close an SFT/MFT Entry
;
;       MFTclose(SFT)
;
;       MFTclose removes the SFT entry from the MFT structure.  If this was
;       the last SFT for the particular file the file's entry is also removed
;       from the MFT structure.
;
;       Note that the SFT refcount has not yet been decremented by our caller.
;       A refcount of 1 means that the SFT is going idle
;
;       ENTRY   (ES:DI) points to an SFT structure
;               (DS) = (CS) = DOSGroup
;       EXIT    NONE
;       USES    ALL but DS, ES:DI

Procedure   MFTclose,NEAR
    ASSUME  ES:NOTHING
        mov     ax,es:[di].sf_MFT
        or      ax,ax
        jz      mcl10
        push    ds                      ; No entry for it, ignore ('C' clear)
        push    es
        push    di                      ; preserve regs
        call    CSL
    ASSUME  DS:NOTHING                  ; clear SFT locks
        mov     al,es:[di].sf_ref_count ; (ax) = ref count
        dec     al
        jnz     mcl9
        call    RSC                     ; ref count not 1 - don't dechain
        jnz     mcl9                    ; remove SFT from chain
        call    RMN                     ; not the last SFT for this name
                                        ; remove name record
mcl9:   pop     di
        pop     es                      ; restore regs for exit
        pop     ds

mcl10:  return

EndProc     MFTclose
```

```
BREAK <Set_Block - Try to set a lock>

;**     set_block - set byte range lock on a file
;
;       set_block sets a lock on a specified range of a file.
;       An error is returned if the lock conflicts with another.
;       Locks are cleared via clr_block.
;
;       ENTRY   (ES:DI) = SFT address
;               (CX:DX) = offset of area
;               (SI:AX) = length of area
;               Uid = 16-bit user id of issuer
;               Pid = 16-bit process id of issuer
;               (CS) = (DS) = DOSGroup
;
;       EXIT    'C' clear if no error
;               'C' set if error
;                 (ax) = error code
;                        ('error_lock_violation' if conflicting locks)
;       USES    ALL but DS

Procedure set_block,NEAR
    ASSUME ES:NOTHING

        DEBUG   1,8,<Set_Block 1:8 - offset $x:$x len $x:$x... >,<cx,dx,si,ax>
        push    ds                      ; preserve (ds)
        push    di
        call    clp                     ; do common setup code
    ASSUME DS:NOTHING
        jc      sblk9                   ; error - lock conflict
;
;       Its ok to set this lock.  Get a free block
;       and fill it in
;
;       (ds:si) = pointer to name record
;       (ax:bx) = fba lock area
;       (cx:dx) = lba lock area
;       (es:TOS) = SFT address
;       (TOS+1) = saved (ds)

        call    OFL                     ; (ds:di) = ptr to new, orphan lock record
        jc      sblk9                   ; no more space
        mov     WORD PTR [di].rlr_sptr+2,es ; store SFT segment
        pop     WORD PTR [di].rlr_sptr  ; store SFT offset
        mov     [di].rlr_fba+2,ax
        mov     [di].rlr_fba,bx         ; store lock range
        mov     [di].rlr_lba+2,cx
        mov     [di].rlr_lba,dx
```



```
;       add to front of chain
;
;       (ds:si) = fwa MFT name record

        mov     ax,[si].mft_lptr
        mov     [di].rlr_next,ax
        mov     [si].mft_lptr,di
        clc                             ; no error

;       Exit -
;
;       'C' set
;       (TOS) = saved (DS)

sblk8:  pop     ds
        ret

;       Exit -
;
;       'C' set
;       (TOS) = garbage
;       (TOS+1) = saved (ds)

sblk9:  pop     bx
        DEBUG   1,8,< CONFLICT - UNSET\n>,<>
        pop     ds
        ret

EndProc set_block
```

```
BREAK <Clr_Block - Try to clear a lock>

        clr_block - clear byte range lock on a file

;**     clr_block clears a lock on a specified range of a file.
        Locks are set via set_block.

        ENTRY   (ES:DI) = SFT address
                (CX:DX) = offset of area
                (SI:AX) = length of area
                Uid = 16-bit user id of issuer
                Pid = 16-bit process id of issuer
                (CS) = (DS) = DOSGroup

        EXIT    'C' clear if no error
                'C' set if error
                  (ax) = error code
                         ('error_lock_violation' if conflicting locks
                          or range does not exactly match previous lock)

        USES    ALL but DS

Procedure clr_block,NEAR
    ASSUME ES:NOTHING

        DEBUG   1,8,<Clr_Block 1:8 - offset $x:$x len $x:$x... >,<cx,dx,si,ax>
        push    ds                      ; preserve (DS)
        push    di
        call    clp                     ; do common setup code
    ASSUME  DS:NOTHING
        pop     bp                      ; ES:BP points to sft.
        jnc     cblk8                   ; error - lock dosent exist
        jnz     cblk8                   ; error - range not given correctly

        DEBUG   1,8,< REMOVED\n>,<>

        We've got the lock

        (DS:di) = address of pointer (offset) to previous lock record
        (es:BP) = sft address

        Now comes the tricky part.  Is the lock for us?  Does the lock
        match the lock that was given us?  If not, then error.

        mov     si,[di]                 ; (DS:SI) = address of lock record
        cmp     word ptr [si].rlr_sptr,bp
        jnz     cblk8                   ; do a 32-bit compare
        mov     bp,es
        cmp     word ptr [si].rlr_sptr+2,bp
        jnz     cblk8
```



```
;       The locks match the proper open invocation.  Unchain the lock

        mov     ax,[si].rlr_next
        mov     [di],ax                 ; chain it out
;       put defunct lock record on the free chain
;
;       (ds:si) = address of freed lock rec

cblk4:  mov     ax,Frelock
        mov     [si].rlr_next,ax
        mov     Frelock,si              ; stuff on front of chain
        clc
        jmp     short cblk9

;       error - lock range not right

cblk8:  mov     ax,error_lock_violation
        DEBUG   1,8,< NOT REMOVED\n>,<>
        stc

;       Exit
;
;       'C' set correctly
;       (TOS) = Saved DS

cblk9:  pop     ds
        ret

EndProc clr_block
```

```
BREAK <Chk_Block - See if the specified I/O violates locks>
;**
;       chk_block - check range lock on a file
;
;       chk_block is called to interogate the lock status of a region
;       of a file.
;
;       NOTE:   This routine is called for every disk I/O operation and
;               MUST BE FAST
;
;       ENTRY   (ES:DI) points to an SFT structure
;               (CX) is the number of bytes being read or
;                   written
;               BytPos is a long (low first) offset into
;                   the file of the I/O
;               Uid = 16-bit user id of issuer
;               Pid = 16-bit process id of issuer
;               (SS) = DOSGroup
;
;       EXIT    'C' clear if no error
;               'C' set if the region is locked by a
;                   process/user other than the issuer.
;
;       USES    ALL  but ES,DI,CX,DS

Procedure   chk_block,NEAR

ASSUME  DS:NOTHING,ES:NOTHING

        push    es
        push    di                      ; preserve regs
        push    cx
        push    ds
        mov     si,CS
        mov     ds,si                   ; address MFT
        mov     si,es:[di].sf_MFT       ; (DS:SI) = address of MFT record
        test    [si].mft_lptr,-1
        jz      chkb9                   ; no locks, no need to go any further
        sub     cx,1                    ; (cx) = count-1
        cmc
        jnc     chkb9                   ; 0 bytes... thats OK!
        mov     ax,WORD PTR BytPos+2
        mov     bx,WORD PTR BytPos      ; (ax:bx) = offset
        sub     dx,cx
        add     cx,cx
        adc     dx,bx
        sub     cx,ax                   ; (cx:dx) = lba of lock area
        call    IFS_IGNORE_LOCKS        ; ignore own locks
        mov     ax,error_lock_violation ; assume error
```



```
;       exit
;       'C' and (ax) setup

chkb9:
        pop     ds
        pop     cx                      ; restore regs
        pop     di
        pop     es
        ret

EndProc chk_block
```